STOCHASTICALLY BASED THREAD BUDGET OVERRUN HANDLING SYSTEM AND METHOD

FIELD OF THE INVENTION

[0001] The present invention generally relates to real-time computer operating systems and, more particularly, to time-partitioned, real-time computer operating systems which schedule and execute multi-threaded applications.

BACKGROUND OF THE INVENTION

[0002] Real-time computer operating systems can be configured to supervise and execute multiple application threads concurrently. These operating systems permit the programming code to be organized such that, conceptually, these multiple application threads are being executed simultaneously. In reality, however, the operating system is switching between each of the threads, in accordance with a programmed schedule.

[0003] In addition, some real-time computer systems may be configured to provide both space partitioning and time partitioning. Partitioning refers generally to separation of resources used by various application threads in a system, such that the resources utilized by a one thread are protected from corruption or usurping by another thread. Space partitioning refers to the protection of resources such as, for example, memory and I/O devices. Time partitioning refers to assurance that a given thread will have access to a predetermined amount of CPU time, regardless of the attempted behavior of any other thread.

[0004] One example of a real-time system that may be both space- and time-partitioned, is an aircraft avionics system. An avionics system may include application threads that are used to monitor and control functions of varying criticality, and that may execute using the same processing assets, such as a single central processing unit (CPU) or the same memory (or portions of memory). Thus, the application schedule allocates CPU time to each of the

various application threads, and ensures that the application threads used to monitor and/or control less critical functions do not prevent application threads used to monitor and/or control higher criticality functions from executing in a timely manner.

[0005] The application schedule allocates CPU time based on a budgeted time and a rate for each application thread. More particularly, each application thread has a particular activation time budget and activation period. An application thread may be repeatedly activated at its particular activation period, and for its particular amount of budgeted activation time during each activation period. For example, a first application thread may have a budgeted activation time of 3 units of time and an activation period of 5 units of time, and a second application thread may have a budgeted activation time of 4 units of time and an activation period of 10 units of time (e.g., the first application thread has an activation frequency that is twice that of the second application thread). Thus, when each application thread is operating within its assigned budget, the first application thread may be activated for up to 3 units of time every 5 units of time, and the second application may be activated for up to 4 units of time every 10 units of time.

[0006] The activation time budget of each application thread typically includes at least an execution time and a pad time. The execution time is the amount of time needed for the application thread to perform its intended function. The pad time is included as part of the activation time budget to allow for instances in which an application thread may attempt to overrun its execution time budget. The magnitude of this pad time is determined based on postulated or measured worst-case response conditions, and is included in the activation budget of each application thread. Thus, the overall activation time budget for an application is the sum of the worst-case activation time budgets for each of the individual threads of the application.

[0007] Although the above-described method of handling application thread activation time budget overruns provides for some design conservatism, it may also result in overall system underutilization. This is because the magnitude of the pad time is fairly pessimistic compared to execution time overrun values actually experienced, and the overall activation time budget for each application is subtracted from the overall system utilization time pool.

[0008] Accordingly, there is a need for a system and method for handling application thread activation time budget overruns that increases system utilization capabilities as

compared to present systems and methods. Furthermore, other desirable features and characteristics will become apparent from the subsequent detailed description and the appended claims, taken in conjunction with the accompanying drawings and this background section.

BRIEF SUMMARY OF THE INVENTION

[0009] In one embodiment, and by way of example only, a method of apportioning additional thread activation time to computer application threads that experience activation time budget overruns includes setting a variable to a predetermined system pad time value. An application thread is activated for an actual activation time. The application thread's actual activation time is compared with its activation time budget. If the actual thread activation time exceeds the thread activation time budget, the system pad time value is adjusted to thereby obtain an updated pad time value.

[0010] In another exemplary embodiment, a system for apportioning additional thread activation time to computer application threads that experience activation time budget overruns includes a memory and a processor. The memory is adapted to store at least a system pad time value therein. The processor is in operable communication with the memory and is operable to (i) activate one or more application threads for an actual activation time; (ii) compare the application thread's actual activation time with its activation time budget; and (iii) if the actual thread activation time exceeds the thread activation time budget, adjust the system pad time value to obtain an updated pad time value.

[0011] In yet another exemplary embodiment, a computer readable medium containing computer executable code for instructing a computer, which is configured to apportion additional thread activation time to computer application threads that experience activation time budget overruns, to perform the steps of setting a variable to a predetermined system pad time value. An application thread is activated for an actual activation time. The application thread's actual activation time is compared with its activation time budget. If the actual thread activation time exceeds the thread activation time budget, the system pad time value is adjusted to thereby obtain an updated pad time value.

BRIEF DESCRIPTION OF THE DRAWINGS

[0012] The present invention will hereinafter be described in conjunction with the following drawing figures, wherein like numerals denote like elements, and wherein:

[0013] FIG. 1 is a functional block diagram of an exemplary system that may implement an embodiment of the present invention;

[0014] FIG. 2 is a diagram illustrating an exemplary activation time budget and its components;

[0015] FIG. 3 is a diagram illustrating the activation time budgets of the application threads for an exemplary two thread application;

[0016] FIG. 4 depicts a flowchart showing an exemplary application thread scheduling process that may implement the present invention; and

[0017] FIG. 5 is a timing diagram depicting the scheduling of the application threads shown in FIG. 3, in accordance with the exemplary process shown in FIG. 4.

DETAILED DESCRIPTION OF A PREFERRED EMBODIMENT

[0018] The following detailed description of the invention is merely exemplary in nature and is not intended to limit the invention or the application and uses of the invention. Furthermore, there is no intention to be bound by any theory presented in the preceding background of the invention or the following detailed description of the invention.

[0019] A functional block diagram of an exemplary system that may implement the present invention is illustrated in FIG. 1. The system 100, which may, for example, form part of an aircraft avionics system, includes a processor 102, and a memory 104 in communication with the processor 102. The processor 102 may include on-board RAM (random access memory) 106, and on-board ROM (read only memory) 108. The processor 102 may be any one of numerous known general purpose microprocessors or an application specific processor that operates in response to program instructions. Such program

instructions may be stored in either or both the RAM 106 and the ROM 108. For example, the operating system software may be stored in the ROM 108, whereas various operating mode software routines and various operational parameters may be stored in the RAM 106. It will be appreciated that this is merely exemplary of one scheme for storing operating software and software routines, and that various other storage schemes may be implemented.

[0020] The memory 104, which is in communication with the processor 102 via any one of numerous known data communication bus protocols, may also be used to store some or all of the operating system software or software routines. The memory 104 may also be used to store one or more parameters and variables used by the operating system software and software routines. It will be appreciated that the memory 104 may be a separate circuit component, as it is depicted in FIG. 1, or it may be formed integrally with the processor 102 or some other component in the system 100 into which it is installed.

[0021] The operation of the system 100 is controlled, at least in part, by an operating system that implements time-partitioned, real-time system operation. To do so, the operating system, among other things, implements an application schedule that multiplexes access time to the processor 102 amongst multiple application threads based on each application thread's activation time budget and its rate (or period), which will now be discussed in more detail.

[0022] As was previously noted, each application thread is assigned an activation time budget. An exemplary activation time budget 200 for a single application thread is illustrated in FIG. 2, and includes three major components, a context switch-in time 202, an execution time 204, and a pad time 206. The context switch-in time 202 is the time it takes for the operating system to make an application thread ready for execution. A detailed understanding of the context switch-in time 202 is not necessary to a complete understanding of the present invention. Therefore, it will not be further described.

[0023] The execution time 204, as its name connotes, is the time needed for an application thread to actually perform its intended function. The pad time 206 includes at least the amount of time it takes for the operating system to conduct an application thread context switch-out. However, it is noted that the pad time 206 may also include some pad time in addition to the context switch-out time. Nonetheless, its magnitude is less than what it

would be if it were based on postulated worst-case response conditions. This is because the present embodiment of the operating system allows for a "system pad time," to be set aside. This system pad time, which is subtracted from the overall system time budget, is available to all the application threads executing on the system 100. Thus, the pad time 206 portion of each application thread's activation time budget 200 may be significantly reduced, as compared to postulated worst-case conditions. If an application thread does attempt to exceed its execution time 204, and its budgeted pad time 206 is insufficient to cover the time needed to complete the execution, the application thread may draw the needed time from the system pad time. It will be appreciated that an application thread may exceed its budgeted activation time for various reasons. For example, the system 100 may be in a critical section when the thread timer expires (e.g., at the end of 204), and/or the path through the operating system code that gets out of the current thread may take longer than expected due to, for example, cache effects. It will additionally be appreciated that each time an application thread draws time from the system pad time during a period, the amount of system pad time that other application threads that execute may draw from during the same period is reduced. A more detailed description of this particular process will now be described in more detail. In doing so, reference should be made to FIGS. 3-5 in combination, which are used to illustrate how an operating system implements an application schedule that includes an embodiment of the present invention.

[0024] Referring first to FIG. 3, which shows the time budgets for each of the exemplary application threads used to describe the above-noted process, it is seen that the exemplary application 300 includes only two threads, a first application thread 302, and a second application thread 304. In this example, the first application thread 302 has an activation time budget of 100 units, and the second 304 application thread has activation time budget of 200 units. For ease of explanation, both the first 302 and second 304 threads execute during the same base period. As is shown, the activation time budgets for each of the application threads 302, 304 include a pad time of 10 units. As was noted above, for clarity, the above-mentioned context switch-in time 202 is not illustrated in FIG. 3.

[0025] An exemplary embodiment of a scheduling process 400, which is implemented by the operating system to multiplex processor access time for each of the application threads 302, 304, is shown in FIG. 4, and will now be described in conjunction with the timing diagram shown in FIG. 5. It should be understood that the parenthetical references in the

following description correspond to the reference numerals associated with the flowchart blocks shown in FIG. 4.

[0026] When the application schedule process 400 begins, it initializes various parameters (402). Included among these parameters is the system pad time value (SYS_PAD_TIME). It will be appreciated that the process 400 could be implemented with more or less variables than what is explicitly shown and described herein. However, for clarity and ease of explanation of the present embodiment, only these two parameters are shown and described.

[0027] No matter the particular number of variables to be initialized, in the present embodiment, the system pad time value (SYS_PAD_TIME) is set to an initial value ($SYS\_PAD\_TIME_{init}$). This initial time value is set aside for the above-described system "pad time," and is subtracted from the overall system time budget. It will be appreciated that the initial value to which this parameter is set may vary, and may be determined in any one of numerous ways. For example, the initial value may be a fixed number that is predetermined and input by system designers, it may be a fixed or variable value that is determined and input by system users, or it may be a variable value is recomputed at each period or some other time frame, just to name a few non-limiting examples. In a particular preferred embodiment, the system pad time value (SYS_PAD_TIME) is initially set to a predetermined initial value ($SYS\_PAD\_TIME_{init}$) by, for example, a system integrator, and its particular value is preferably determined based on empirical data. No matter the particular value or manner to which it is set, this particular amount of time, as will be described more fully below, is available to the application during each period. For the example shown in FIG. 5, the system pad time value ($SYS\_PAD\_TIME_{init}$) is set to a value of 30 units.

[0028] After the system pad time value is initialized, the period is commenced (404). At this point, the active thread number (N) is set to "1," which corresponds to the first thread to execute during the period. Thereafter, the activation time budget (TIME_BUDGET(N)) for the active thread (N) is set (406). The value to which the activation time budget (TIME_BUDGET(N)) is set is the active thread's entire activation time budget. As was noted above, the active thread (e.g., N=1) is the first application thread 302. Hence, the activation time budget for the first thread 302 (TIME_BUDGET(1)) is set to 100 units.

[0029] Before the active thread (N) actually executes, the operating system subtracts the active thread's pad time value from its activation budget and stores the subtracted pad time value in memory (408). An application thread timer (THREAD_TMR(N)) for the active thread (N) is then set to the time that remains after the pad time is subtracted from the active thread budget. The thread timer for each application thread 302, 304, which may be implemented in hardware, software, or firmware, is preferably a decrementing type of timer that may decrement below zero. Hence, in the present example, in which the first thread 302 is the active thread (e.g., N=1), a value of 90 units is stored in memory, and the thread timer (THREAD_TMR(1)) is set to 90 units. Thereafter, at time t=0 in FIG. 5, the thread timer for the first application thread ((THREAD_TMR(1)) begins decrementing (410), and the first application thread 302 begins executing (412). It will be appreciated that the thread timers (THREAD_TMR(N)) could also be implemented as incrementing type of timers.

[0030] Once the active thread (N) begins executing (412), it continues doing so until its execution is complete or its thread timer (THREAD_TMR(N)) reaches zero (414). It will be appreciated that the process 400 could include additional and/or other events that would terminate, at least momentarily, the execution of the active thread (N). However, these additional or other events are not needed to understand the present invention and will, therefore, not be further discussed. In either case, once thread execution stops, its thread timer (THREAD_TMR(N)) continues decrementing until the operating system is ready to make the next thread the active thread (418). In the context of the present example, and as shown in FIG. 5, the first application thread 302 does not complete its execution within the expected time, and its thread timer (THREAD_TMR(1)) reaches zero at time t=90. However, for the reasons previously delineated, the operating system is not ready to make the second thread 304 active until time t=110. At that point in time, the first thread's thread timer (THREAD_TMR(1)) is stopped. As can be seen, the first application thread 302 exceeded its budgeted activation time (e.g., 100 units) by 10 units.

[0031] After the thread timer (THREAD_TMR(N)) of the active thread (THREAD(N)) stops (416), the active application thread's pad time value 206 is retrieved from memory and is added back to the time remaining on the active thread's thread timer (418). This resultant sum (SUM(N)) is then checked to determine whether it is negative (420). If it is negative, this indicates that the actual activation time of the active thread (THREAD (N)) exceeded its activation time budget and the resultant sum (SUM(N)) value is added to the system pad time value (SYS_PAD_TIME), and the resultant sum is then re-zeroed (SUM(N)=0) (422).

Adding this negative sum value (SUM(N)) to the system pad time value (SYS_PAD_TIME) decrements the "system pad time." Conversely, if the resultant sum (SUM(N)) is non-negative, this indicates that the actual activation time of the active thread (THREAD(N)) was within its activation time budget, and no adjustment to the system pad time value is needed. In the context of the present example, and as shown in FIG. 5, at time t=110, the pad time of the first application thread 302 (e.g., 10 units) is added back to the active thread timer (THREAD_TMR(1)). In this instance, the active thread timer (THREAD_TMR(1)) decremented to a value of -20 units. Thus, the resultant sum (SUM(1)) is a negative value of -10 units, and the system pad time value is adjusted to a value of 20 units. Therefore, 20 units of system pad time remain in the system "pad pool" for the second application thread 304 to draw from, if needed, during the present base period.

[0032] Once the determination of whether an adjustment to the system pad time value (SYS_PAD_TIME) is needed (420) and, if so, the adjustment is made (422), the system pad time value is checked to determine whether it is negative or non-negative (424). If the value is negative, an appropriate system response such as, for example, a fatal system error, will be executed (426). However, if the value is non-negative, no further action is taken. Instead, the active thread number (N) is incremented (428), and the operating system determines whether the thread that just executed is the last thread of the given period (430). If thread that just executed is not the last thread of the period, the operating system executes a context switch-in to the next active thread (N). Thereafter, process steps (406-428) are repeated. However, if the thread that just executed is the last one for the present period, then a new period is started, and process steps (404-428) are repeated. For the present example, the adjusted system pad time value (SYS_PAD_TIME) is non-negative (e.g., 20 units). Thus, the active thread number (N) is incremented to "2," making the second application thread 304 the active thread, and the previously described steps (406-428) are executed.

[0033] As a result, the second application thread's activation time budget (TIME_BUDGET(2)) is set to 200 units (406), and its pad time (e.g., 10 units) is subtracted from its activation time budget, and is stored in memory (408). Then, at approximately time t=110 in FIG. 5, the second application thread's thread timer begins to decrement (410), and the second application thread 304 begins executing (412). It will be appreciated that the second application thread 304 may begin executing slightly later than t=110, due to the time

it takes to read, write, and store various data, and also due to the context switch-in time, described above.

[0034] It is seen in FIG. 5 that the second application thread 304 continues executing until time t=250, at which point its execution is complete (414). In addition, the operating system is ready to make the next thread active. Thus, the second thread's thread timer THREAD_TMR(2)) is stopped (416) with a value of 40 units remaining on the timer, and the pad time of the second application thread 304 (e.g., 10 units) is added back to the active thread timer (THREAD_TMR(2)) (418). In this instance, the resultant sum (SUM(2)) is non-negative (e.g., 50 units) (420). Because the resultant sum (SUM(2)) is non-negative (420), no adjustment is made to the system pad time value (SYS_PAD_TIME). Since both application threads 302, 304 have executed in the period (428), the process illustrated in FIGS. 4 and 5 repeats for the exemplary application.

[0035] It is noted that the example described above and illustrated in FIG. 5 did not result in the system overhead value (SYS_OVRHD) being adjusted to a negative value. However, as was described above and as is shown in FIG. 4, if the system pad time value (SYS_PAD_TIME) is adjusted to a negative value during a period, an appropriate system response such as, for example, a fatal system error, will be executed (426). This may result in the system, or at least a portion of the system, being halted. Depending on the system, it may also result in some type of external annunciation to make operational personnel aware of the error. Moreover, if the affected process is being run on a multi-channel, redundant system, operation may switch to one of the redundant channels within the system.

[0036] In the above-described exemplary process 400, the thread timers are decremented, and the thread level pad time that is subtracted from a thread's activation time budget is added back to the thread timer upon completion of the active thread's execution. It will be appreciated that this methodology is merely exemplary of a particular preferred embodiment and that the thread timers, as was previously mentioned, could increment while the associated threads are executing, and the pad time could be subtracted from the timer upon completion of the active thread's execution.

[0037] While at least one exemplary embodiment has been presented in the foregoing detailed description of the invention, it should be appreciated that a vast number of variations exist. It should also be appreciated that the exemplary embodiment or exemplary

embodiments are only examples, and are not intended to limit the scope, applicability, or configuration of the invention in any way. Rather, the foregoing detailed description will provide those skilled in the art with a convenient road map for implementing an exemplary embodiment of the invention. It being understood that various changes may be made in the function and arrangement of elements described in an exemplary embodiment without departing from the scope of the invention as set forth in the appended claims.